Fault-tolerant Programs as Multi-state Systems

Gregory Levitin
The Israel Electric Corporation
Reliability Department
Israel
levitin@iec.co.il

Abstract

The paper suggests a model of fault tolerant software system and demonstrates that this system has multi-state nature when the execution time is considered to be the measure of its performance. It presents an algorithm for evaluating the distribution of the system execution time and formulates optimization problems of software version sequencing and choice.

1. Introduction

Software failures are caused by errors made in various phases of program development. When the software reliability is of critical importance, special programming techniques are used in order to achieve its fault tolerance. Two of the best-known fault-tolerant software design methods are N-version programming (NVP) and recovery block scheme (RBS). Both methods are based on the redundancy of software modules (functionally equivalent but independently developed) and the assumption that coincident failures of modules are rare (Teng and Pham 2003). The fault tolerance usually requires additional resources and results in performance penalties (particularly with regard to computation time), which constitutes a tradeoff between software performance and reliability.

The NVP approach presumes the execution of N functionally equivalent software modules (called versions) that receive the same input and send their outputs to a voter, which is aimed at determining the system output. The voter produces an output if at least M out of N outputs agree (it is presumed that the probability that M wrong outputs agree is negligibly small). Otherwise, the system fails. Usually majority voting is used in which N is odd and $M=(N+1)/2$.

In some applications, the available computational resources do not allow all of the versions to be executed simultaneously. In these cases, the versions are executed according to some predefined sequence and the program execution terminates either when M versions produce the same output (success) or when after the execution of all the N versions the number of equivalent outputs is less than M (failure). The entire program execution time is a random variable depending on the parameters of the versions (execution time and reliability), and on the number of versions that can be executed simultaneously.

In the RBS approach after execution of each version, its output is tested by an acceptance test block (ATB). If the ATB accepts the version output, the process is terminated and the version output becomes the output of the entire system. If all N versions do not produce the accepted output, the system fails. If the computational resources allow simultaneous execution of several versions, the versions are executed according to some predefined sequence and the entire program terminates either when one of versions produces the output accepted by the ATB (success) or when after the execution of all the N versions no output is accepted by the ATB (failure). If the acceptance test time is included into the execution time of each version, the RBS performance model becomes identical to the performance model of the NVP with $M=1$.

The fault-tolerant programs can be considered as multi-state systems with the system performance defined as its total execution time. Estimating the effect of the fault-tolerant programming on system performance is especially important in safety critical real-time computer applications.
Since the performance of fault-tolerant programs depends on availability of computational resources, the impact of hardware availability should also be taken into account when the system reliability is evaluated.
It can be seen that the fault-tolerant system performance and reliability depend on characteristics of software versions as well as on the sequence of their execution.
The paper presents algorithms for reliability and performance evaluation for software and hardware-software fault-tolerant systems. It also formulates the problems of optimal version sequencing and optimal version choice (structure optimization) in such systems and suggests algorithms for solving these problems.

2. Model of System with Fault-Tolerant Software Components

A software system consists of C components. Each component performs a subtask and the sequential execution of the components performs a major task.
It is assumed that N_c functionally equivalent versions are available for each component c. Each version i has an estimated reliability r_{ci} and constant execution time τ_{ci}. Failures of versions for each component are statistically independent as well as the total failures of the different components.
The software versions in each component c run on parallel hardware units. The total number of units is H_c. The units are s-independent and identical. The availability of each unit is A_c. The number of units available at the moment h_c determines the amount of available computational resources and, therefore, the number of versions that can be executed simultaneously $L_c(h_c)$. No hardware unit can change its state during the software execution.
The versions of each component c start their execution in accordance with a predetermined ordered list. L_c first versions from the list start their execution simultaneously (at time 0). If the number of terminated versions is less than M_c, after termination of each version a new version from the list starts its execution immediately. If the number of terminated versions is not less than M_c, after termination of each version the voter compares the outputs. If M_c outputs are identical, the component terminates its execution (terminating all the versions that are still executed), otherwise a new version from the list is executed immediately. If after termination of N_c versions the number of identical outputs is less than M_c the component and the entire system fail.
In the case of component success, the time of the entire component execution is equal to the termination time of the version that has produced the M_c-th correct output (in most cases the time needed by the voter to make the decision can be neglected). It can be seen that the component execution time is a random variable depending on the reliability and the execution time of the component versions and on the availability of the hardware units.
The examples of the task execution time diagrams corresponding to a component with N_c=3, M_c=2 for a given sequence of versions execution (the versions are numbered according to this sequence) and different values of L_c are presented in Fig. 1.
The sum of the random execution times of each component gives the random task execution time for the entire system. In order to estimate both the system's reliability and its performance, different measures can be used depending on the application.
In applications where the execution time of each task is of critical importance, the system reliability $R(T^*)$ is defined as a probability that the correct output is produced in time less than T^*.
In applications where the average system productivity (the number of executed tasks) over a fixed mission time is of interest, the system reliability is defined as the probability that it produces correct outputs without respect to the total execution time, (this index can be referred to as $R(\infty)$) while the

conditional expected system execution time E is considered to be a measure of its performance. This index determines the system expected execution time given that the system does not fail.

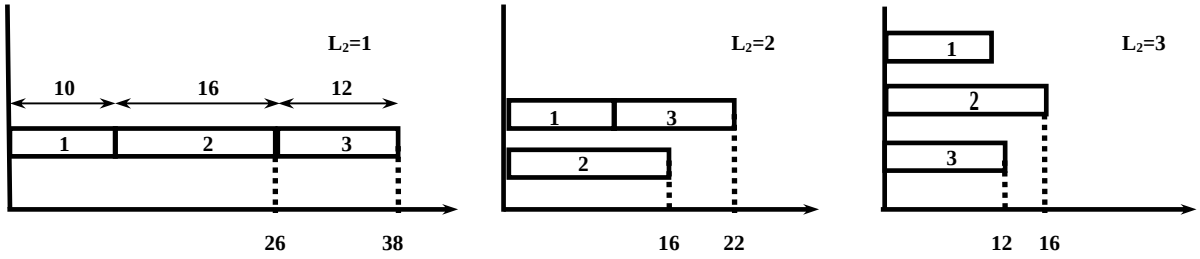

Figure 1: Time diagrams for software component with N_c=3, M_c=2

3. Fault-Tolerant System Optimization Problems

The method for obtaining the pmf of the random system task execution time is based on the Universal Generating Function technique (Ushakov (1987), Lisnianski and Levitin (2003)). This corresponding algorithm (Levitin (2004a)) allows one to evaluate in a short time the following system performance indices: the system reliability defined as the probability that it produces the correct output in less time than a specified value T^*; the probability that the correct output is obtained without respect to the execution time and the expected system execution time on condition that the system does not fail.

Having the fast algorithm for the fault tolerant system performance evaluation one can effectively solve optimization problems aimed at improving the system performance.

Since the sequence of version execution affects the probabilistic distribution of the system execution time, two different optimization problems can be formulated in which the sequences maximizing the system reliability $R(T^*)$ or minimizing its conditional expected execution time E are to be determined (Levitin (2004)b). In these problems the set of versions to be executed is determined for each software component. The optimal version sequencing can improve the system performance without additional expenses.

Another optimization problem is how to choose the optimal structure of software components from the list of available versions. Consider a set of J_c different versions available for each component c. Each version j ($1 \le j \le J_c$) is characterized by its reliability $r_c(j)$, execution time $\tau_c(j)$ and cost $s_c(j)$. The permutation $\boldsymbol{z}_c$ of J_c different integer numbers ranging from 1 to J_c determines the order of the version that can be used in component c. Let h_{cj}=1 if the version j is chosen to be included into component c and h_{cj}=0 otherwise. The vector $\boldsymbol{h}_c = \{h_{c1},...,h_{cJ_c}\}$ determines the subset of chosen versions for component c.

Having the vectors $\boldsymbol{z}_c$ and $\boldsymbol{h}_c$ one can determine the execution order $\boldsymbol{v}_c$ of the chosen versions by removing from $\boldsymbol{z}_c$ any number k for which h_{ck}=0. The total number of versions in component c (equal to the length of vector $\boldsymbol{v}_c$) is determined as

$$N_c = \sum_{j=1}^{J_c} h_{cj}.$$

The system structure optimization problem can now be formulated as follows: find vectors $\boldsymbol{v}_c$ for $1 \le c \le C$ that maximize $R(T^*)$ subject to cost constraint:

$$S = \sum_{c=1}^{C} \sum_{j \in \boldsymbol{v}_c} s_c(j) \le S^*.$$

4. Example of Software System Optimization

Consider a fault-tolerant system consisting of four software components running on single reliable hardware unit. The parameters of versions that can be used in these components are presented in Table 1. This table contains the values of M_c and L_c for each component and the cost, reliability and execution time for each version.

For T^*=300 four different solutions were obtained for different cost constraints. These solutions are presented in Table 2. The tables contain the system cost and reliability for each solution, the expected conditional execution time, minimal and maximal possible system execution times and the corresponding sequences of the chosen versions.

Table 1: Parameters of system components for the numerical example

No of component	*M*	*L*		versions							
				1	2	3	4	5	6	7	8
1	1	1	τ	17	10	20	32	30	75	-	-
			r	0.71	0.85	0.85	0.89	0.95	0.98	-	-
			c	5	15	7	8	12	6	-	-
2	2	2	τ	28	55	35	55	58	-	-	-
			r	0.82	0.82	0.88	0.90	0.93	-	-	-
			c	11	8	18	10	16	-	-	-
3	3	4	τ	17	20	38	38	48	50	41	63
			r	0.80	0.80	0.86	0.90	0.90	0.94	0.98	0.98
			c	4	3	4	6	5	4	9	6
4	2	1	τ	17	10	20	32	-	-	-	-
			r	0.75	0.85	0.93	0.97	-	-	-	-
			c	12	16	17	17	-	-	-	-
5	1	3	τ	30	54	40	65	70	-	-	-
			r	0.70	0.80	0.80	0.80	0.89	-	-	-
			c	5	9	11	7	12	-	-	-

Table 2: Parameters of obtained solutions

S^*	Sequence of versions	*S*	*R*	*E*	t_{min}	t_{max}
160	341|4521|85632|324|413	160	0.951	210.82	188	369
140	53|541|28361|431|51	140	0.889	231.02	208	335
120	6|241|61372|241|31	120	0.813	252.87	240	307
100	4|142|2386|43|41	100	0.672	238.05	219	295

References

Teng, X., Pham H., (2003). Software fault tolerance, in *Reliability Engineering Handbook*, H. Pham (Ed.), Springer, 585-611

Ushakov, I. (1987). A universal generating function. *Soviet Journal Computer Systems Science*, 61-73

Lisnianski A., Levitin, G. (2003). *Multi-state system reliability. Assessment, optimization and applications.* World Scientific.

Levitin, G. (2004a) Reliability and performance analysis for fault-tolerant programs consisting of versions with different characteristics. To appear in *Reliability Engineering and System Safety*.

Levitin, G. (2004b) Optimal version sequencing in fault-tolerant programs. To appear in *Asia-Pacific Journal of Operational Research*.